# ASGN #7 DESIGN

badspeak.txt

cse 13s
tiktok
hole

contains bad word least

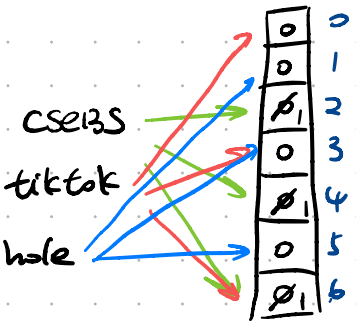
newspeak.txt

python → slow
c → fast
music → joy

new speach trans
old → new

→ static length x growth

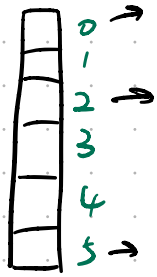
bloom filter:




more index

↳ gonna be a pointer to _(?)

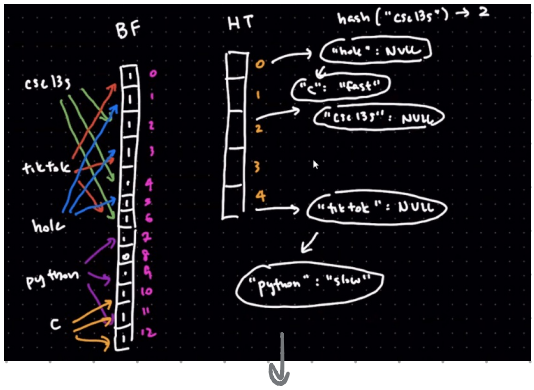
hash table




less

hash(pr:, "cse13s") → 2
hash(sec, "cse13s") → 4
hash(ter, "cse13s") → 6
↳ need to hash 3 times

hash("cse13s") → 2




much slower to ask compare to BF.

input: "i use python in cse13s"

↓

how to check if word in BF: → check if each 3 hashes are set to 1 in BF (bf_probe)

↓

check hash table
↳ hash(use) ← checking if "use" is in HT
eg. hash(use) → 3
↳ HT[3] is empty
→ use is not bad word

revise = ["python" → "slow"]
bad = ["cse13s"]

check BF
↳ BF ✓
↳ check HT make sure

message.h

regular expression:

+ : 1 or more

* : 0 or more

| : OR

? : optional (0 or 1)

[ ] : symbol set

only can start with _, num, Alph

(a|b)*
aaa or bbb (only a or b)
(ab)*
ababab (a & b)

rm *.txt => rm every ends with "_.txt"

(0|1)+ => binary

0x([a-f0-9])+ => hex

([a-z])+ -([a-z])+ => y_all_ve

Program purpose:

Find bad/old word in the sentence.

If the old speech has a new speech that could match to it => change to new speech

→ There are 3 kind of messages ←

bv.c (only gonna use in BF) similar as code.c in asgn5

- struct BitVector {
    u32 length
    u8 * vector
    }

- creat ( length)
    BV *bv = malloc
    if(bv) {
      bv -> length = length
      for ( i=0; i < length ; i++){
          vector[i]=0
      }
    }
    return.

- bv_delete
    if (* bv && (*bv) -> length != 0) {
      free((*bv) -> vector)
      length=0
    }

- length (bv)
    return bv -> length

- set_bit (bv. i)
    if (i < length) {
      |= 1 << i %8
      return true
    }
    else
      return false

- clr_bit (*bv, i)

```
if (i < length)
    &= ~(1 << i%8)
    return true.
else
    return false
```

AND 0 => lose value.

- get_bit (*bv, i)

```
if (i > length)
    return false
else {
    if ((>> i) & 1 == 1
        return true
    else
        return false
    }
```

- print

```
printf( length)
for ( i<length)
    print  bv->vector
```

bf.c

- struct BF{

```
u64 primary[2]
u64 secondary[2]
u64 tertiary[2]
BV  *filter
}
```

- bf_creat(size)

```
BF *bf = malloc
if (bf){
    pr[0] = hash (pr_lo)
    pr[1] = hash (pr_hi)
    se[0] = hash(se_lo)
    se[1] = hash (se_hi)
    te[0] = hash (te_lo)
```

```
    te[i] = hash(te_hi)
    *filter = bv_creat(size)
}
```

- bf_delete(**bf)

```
if(*bf && (*bf)->filter->length!=0){
    bv_delete
    free(*bf)
    *bf = NULL
}
```

- bf_size

```
return bv_length(bf)
```

- bf_insert

```
hash(primary, old)
bv_set_bit
hash(secondary, old)
bv_set_bit
hash(ter, old)
bv_set_bit
```

- bf_probe

```
hash(pr, old)
hash(se, old)      ) if all 3 get_bit == 1
hash(ter, old)        return true

                   else
                       false
```

- bf_count

```
count=0;
for(i<size){
  if(bf->filter[i] == 1){
      count++;
  }
}
```

- bf_print

need to call bv_print

# node.c

- struct Node {

```
char *oldspeak;
char *newspeak;
Node *left;
Node *right;
}
```

- create( old, new)

```
Node *n = malloc
if (old = null) {
    n -> old = NULL
} else
    n -> old = old
if (new = null) {
    n -> new = NULL
} else
    n -> new = new
n -> left = NULL
n -> right = NULL
```

- delete

```
free (*old)
free (*new)
free (* n -> left)
free (* n -> right)
free (*n)
```

- print

```
if (new = = NULL)
    printf ("%s \n", n -> old );
else
    printf ("%s -> %s\n",  n -> old , n -> new).
```

bst.c

- creat

```
return NULL
```

- delete (root)

```
if (root) {
    delete (root->left)
    delete (root->right)
    node_delete
}
```

// referance : lecture notes.

- max (x,y)

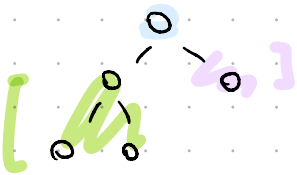
```
return x>y? x:y ;
```

- hight (root)

```
if(root) {
    return 1+ max(bst_height(root->left), bst_height(root->right));
}
return 0;
```

- size (root)

```
if (root == NULL)
    return 0
else
    return (1+ sizenode->left) + size(noode->right));
```



// from ppt

- find (root, old)

```
if (root){
    if (root->old > old)
        return (bst_find (root->left, old)
    else if (root->old < old)
        return bst_find (root->right, old)
}
return root;
```

- insert(root, old, new)

```
if(root)
    if(find(root) == NULL)
        return;
    else if (root->old > old)
        root->left = insert(root->left, old, new)
    else
        root->right = insert(root->right, old, new)
    return root
return node_create(old)
```

- print(root)

```
if(root){
    print(root->left)
    node_print(root)
    print(root->right)
}
```

ht.c

- struct HT{

```
u64 salt[2]
u32 size
Node **trees
```

}

- creat(size)

```
*ht = malloc
salt[0] = bt_lo
salt[1] = ht_hi
ht->size = size
ht->trees = bst_creat()
```

- delete

```
if(ht)
    bst_delete
    free(ht)
```

- size

```
return ht->size
```

- lookup(ht, old)

```
index = hash(old)
if (bst_find(old) != NULL)
    return bst_find(trees[index], old);
else
    return NULL
```

- insert(old, new)

```
index = hash(old)
trees[index] = bst_insert(trees[index], old, new)
```

remember to update root!

- count(ht)

```
u32 count = 0
for (i < ht_size; i++) {
    if (ht->trees[i] != NULL) {
        count++
    }
}
return count
```

- avg_bst_size(ht)

```
double size = 0
for (i < ht_size; i++) {
    size += bst_size(ht)
}
avg_size = size / count
return avg_size
```

- avg_height(ht)

```
height = 0
for (i < ht_size; i++) {
    height += bst_height
}
avg_height = height / count
return avg_height
```

```
-print(ht)
    for( i< ht->size; i++){
        bst_print( )
    }
```

## banhammer.c

default htSize = $2^{16}$
default bfSize = $2^{20}$

command lines
change htSize & bfSize base on the input when there is an input

```
create bf(bfSize)
create ht(htSize)
FILE bfile = fopen("badspeak", "r")
FILE nfile = fopen("newspeak", "r")
old_buf[1024]
new_buf[1024]

while (fscanf(bfil) != EOF) {
    old_word = old_buf
    new_word = NULL
    bf_insert(bf, old)
    ht_insert(ht, old, new)
}

while (fscanf(nfil) != EOF) {
    old_word = old_buf
    new_word = new_buf
    bf_insert(bf, old)
    ht_insert(ht, old, new)
}
```

bfil

nfil

side:
read badspeak.txt

newspeak.txt

fscanf( , "%s\n", buf1)
↑ file,

1024 bytes: buffer.

fscanf( "%s %s\n", buf1, buf2)
↑ file,

```
regex_t re;
    if (regcomp(&re, WORD, REG_EXTENDED)) {
        print error message
        return 1
    }

char *word = NULL;
while (word = next_word(stdin, &re)) != NULL) {
    if (bf_probe(bf, word)) {
        n = look_up(ht, word)
        if (n) {
            if (!have n->newspeck) {
                insert n->oldspeck => old_speck
            }
            else {
                insert n->oldspeck & n->newspeck => new_speck
            }
        }
    }
}

if (old_bst size != 0 && new_bst size != 0) {
    print mixspeck_message
    print (old_bst)
    print (new_bst)
} else if (old_bst size != 0 && new_bst size == 0)
    print badspeck message
    print (old_bst)
} else if (old_bst size == 0 && new_bst size != 0)
    print goodspeck message
    print (new_bst)
}

if (test_s) {
    print ("Avg_BST_size: %.6f\n", avg_bst_size)
    print (".... height: %.6f\n", ... height)
    print avg branches = (double) branches / (double) lookups
        (use %.6f\n)
```

```
        print ht-load  =100 * ((double) ht_count(ht) / (double)ht-size)
              (use  %.6f\n)
        print bf_load = 100 * (double) bf_count (bf) / (double) bf_size)
              (use  %.6f\n)
    }

    clear all
    file close
    return 0;
}
```